Java Subclasses - Complete Notes

What is a Subclass?

A **subclass** is a class that inherits properties and methods from another class (called the parent class or superclass). It's a fundamental concept in inheritance.

Key Terminology:

- **Subclass** = Child class (inherits FROM another class)
- **Superclass** = Parent class (being inherited BY another class)
- **Inheritance** = Process of acquiring properties from parent class

Basic Syntax:

```java
class ParentClass {
    // Parent class members
}

class ChildClass extends ParentClass {
    // Child class inherits from ParentClass
    // Can add new members or override existing ones
}
```

Key Characteristics:

1. Uses `extends` keyword

```java
class Animal {        // Superclass
    // Animal properties and methods
}

class Dog extends Animal {    // Subclass
    // Dog inherits from Animal
}
```

2. "IS-A" Relationship

- Dog **IS-A** Animal
- Car **IS-A** Vehicle

- Student **IS-A** Person

3. Inheritance Rules

Access Modifier	Inherited by Subclass?
public	✓ Yes
protected	✓ Yes
default (package)	✓ Yes (same package)
private	✗ No (not directly accessible)

Complete Example:

java

```java
// Superclass (Parent class)
class Animal {
    protected String name;      // Accessible to subclasses
    protected int age;
    private String species;     // Not directly accessible to subclasses

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Methods that can be inherited
    public void eat() {
        System.out.println(name + " is eating");
    }

    public void sleep() {
        System.out.println(name + " is sleeping");
    }

    public void makeSound() {
        System.out.println(name + " makes a sound");
    }

    // Getter for private variable
    public String getSpecies() {
        return species;
    }

    protected void setSpecies(String species) {
        this.species = species;
    }
}

// Subclass 1 (Child class)
class Dog extends Animal {
    private String breed;       // New property specific to Dog

    // Constructor
    public Dog(String name, int age, String breed) {
        super(name, age);       // Call parent constructor
        this.breed = breed;
        setSpecies("Canine");   // Use protected method from parent
    }
```

```
    // New method specific to Dog
    public void bark() {
        System.out.println(name + " is barking: Woof! Woof!");
    }

    public void wagTail() {
        System.out.println(name + " is wagging tail");
    }

    // Override parent method
    @Override
    public void makeSound() {
        System.out.println(name + " barks: Woof!");
    }

    // Override parent method with additional behavior
    @Override
    public void eat() {
        System.out.println(name + " is eating dog food");
        wagTail();  // Additional behavior
    }

    // Getter for new property
    public String getBreed() {
        return breed;
    }
}

// Subclass 2 (Child class)
class Cat extends Animal {
    private boolean isIndoor;

    public Cat(String name, int age, boolean isIndoor) {
        super(name, age);
        this.isIndoor = isIndoor;
        setSpecies("Feline");
    }

    // New methods specific to Cat
    public void meow() {
        System.out.println(name + " is meowing: Meow! Meow!");
    }

    public void purr() {
        System.out.println(name + " is purring");
    }
```

```
    // Override parent method
    @Override
    public void makeSound() {
        System.out.println(name + " meows: Meow!");
    }

    @Override
    public void sleep() {
        System.out.println(name + " is sleeping " + (isIndoor ? "indoors" : "outdoors"));
    }

    public boolean isIndoor() {
        return isIndoor;
    }
}

// Subclass 3 (Child class)
class Bird extends Animal {
    private boolean canFly;

    public Bird(String name, int age, boolean canFly) {
        super(name, age);
        this.canFly = canFly;
        setSpecies("Avian");
    }

    public void chirp() {
        System.out.println(name + " is chirping");
    }

    public void fly() {
        if (canFly) {
            System.out.println(name + " is flying");
        } else {
            System.out.println(name + " cannot fly");
        }
    }

    @Override
    public void makeSound() {
        System.out.println(name + " chirps: Tweet! Tweet!");
    }
}
```

Using Subclasses:

java

```
public class AnimalDemo {
    public static void main(String[] args) {
        // Create objects of subclasses
        Dog dog = new Dog("Buddy", 3, "Golden Retriever");
        Cat cat = new Cat("Whiskers", 2, true);
        Bird bird = new Bird("Tweety", 1, true);

        System.out.println("=== DOG ACTIONS ===");
        // Using inherited methods
        dog.eat();         // Overridden method
        dog.sleep();       // Inherited method
        dog.makeSound();   // Overridden method

        // Using Dog-specific methods
        dog.bark();
        dog.wagTail();

        System.out.println("Breed: " + dog.getBreed());
        System.out.println("Species: " + dog.getSpecies());

        System.out.println("\n=== CAT ACTIONS ===");
        cat.eat();         // Inherited method
        cat.sleep();       // Overridden method
        cat.makeSound();   // Overridden method

        // Cat-specific methods
        cat.meow();
        cat.purr();

        System.out.println("Indoor cat: " + cat.isIndoor());

        System.out.println("\n=== BIRD ACTIONS ===");
        bird.eat();        // Inherited method
        bird.makeSound();  // Overridden method
        bird.chirp();      // Bird-specific method
        bird.fly();        // Bird-specific method

        // Demonstrating polymorphism
        System.out.println("\n=== POLYMORPHISM DEMO ===");
        Animal[] animals = {dog, cat, bird};

        for (Animal animal : animals) {
            animal.makeSound();  // Calls overridden method for each type
        }
```

```
    }
}
```

What Subclasses Inherit:

✅ Inherited Members:

- **Public methods and variables** - Fully accessible
- **Protected methods and variables** - Accessible within subclass
- **Default methods and variables** - If in same package
- **Constructors** - Can be called using `super()`

❌ Not Inherited:

- **Private methods and variables** - Not directly accessible
- **Constructors** - Not inherited, but can be called
- **Static methods** - Belong to the class, not inherited

Important Concepts:

1. Constructor Chaining:

java

```java
class Parent {
    public Parent(String name) {
        System.out.println("Parent constructor: " + name);
    }
}

class Child extends Parent {
    public Child(String name, int age) {
        super(name);    // Must call parent constructor first
        System.out.println("Child constructor: " + age);
    }
}
```

2. Method Overriding:

java

```java
class Parent {
    public void display() {
        System.out.println("Parent display");
    }
}

class Child extends Parent {
    @Override   // Good practice to use @Override annotation
    public void display() {
        System.out.println("Child display");
    }
}
```

3. super Keyword:

java

```java
class Child extends Parent {
    @Override
    public void display() {
        super.display();    // Call parent's method
        System.out.println("Additional child behavior");
    }
}
```

Benefits of Subclasses:

1. **Code Reusability** - Don't repeat common code
2. **Organized Structure** - Logical hierarchy of classes
3. **Polymorphism** - Treat objects of different types uniformly
4. **Extensibility** - Easy to add new types
5. **Maintenance** - Changes in parent affect all children

Real-World Examples:

Example 1: Vehicle Hierarchy

java

```java
class Vehicle {
    protected String brand;
    protected int maxSpeed;

    public void start() { System.out.println("Vehicle starting"); }
    public void stop() { System.out.println("Vehicle stopping"); }
}

class Car extends Vehicle {
    private int doors;
    public void openTrunk() { System.out.println("Trunk opened"); }
}

class Motorcycle extends Vehicle {
    private boolean hasSidecar;
    public void wheelie() { System.out.println("Doing a wheelie!"); }
}
```

Example 2: Employee Hierarchy

java

```java
class Employee {
    protected String name;
    protected double salary;

    public void work() { System.out.println(name + " is working"); }
}

class Manager extends Employee {
    private int teamSize;
    public void conductMeeting() { System.out.println("Conducting meeting"); }
}

class Developer extends Employee {
    private String programmingLanguage;
    public void writeCode() { System.out.println("Writing code in " + programmingLanguage); }
}
```

Key Points to Remember:

1. **Subclass extends Superclass** using `extends` keyword
2. **Inherits all public and protected members**
3. **Can add new methods and variables**

4. **Can override parent methods**
5. **Represents IS-A relationship**
6. **Use `super()` to call parent constructor**
7. **Use `super.method()` to call parent methods**
8. **Private members are not inherited but can be accessed through public/protected methods**

Common Mistakes to Avoid:

1. **Forgetting to call super() in constructor**
2. **Trying to access private members directly**
3. **Not using @Override annotation**
4. **Creating too deep inheritance hierarchies**
5. **Overriding methods incorrectly**

Subclasses are essential for creating organized, reusable, and maintainable Java code!